Cache

High Performance Computer System (HPCS) Lab

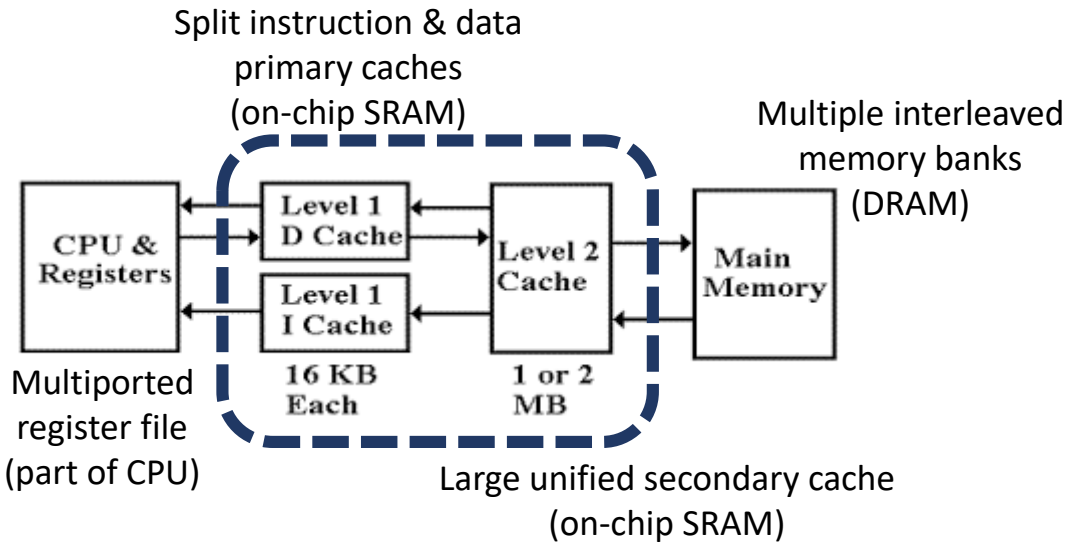
Email (TA): joonsung90@snu.ac.kr

Goals

- Understand cache
- Implement a direct-mapped cache on your pipelined CPU
- Evaluate the speedup achieved by using cache
  - Hit ratio
  - Corresponding speedup (vs. no-cache CPU)

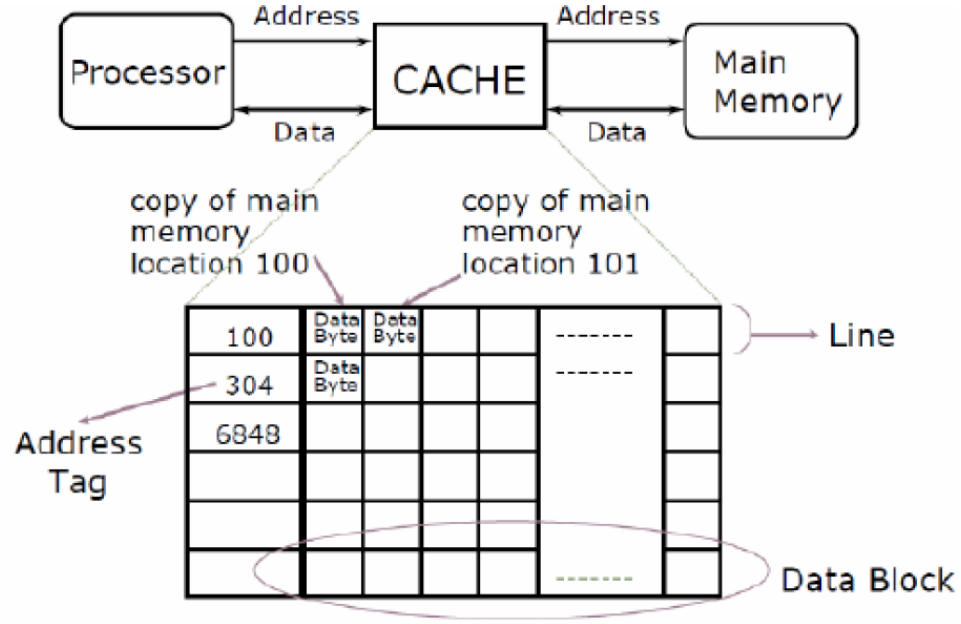
What is Cache?

- Mitigating the performance gap between CPU and memory
  - Memory access ➔ few hundred cycles
  - Cache access ➔ few cycles
- Why does it work?
  - Locality!



Cache: Internal structure

- Tag
  - Detect address conflicts
- Data
  - Fetch by line: exploiting spatial locality



Cache: Associativity

- Associativity
  - Reducing address conflicts
- Direct mapped, n-way, fully associative
  - Tradeoffs exist

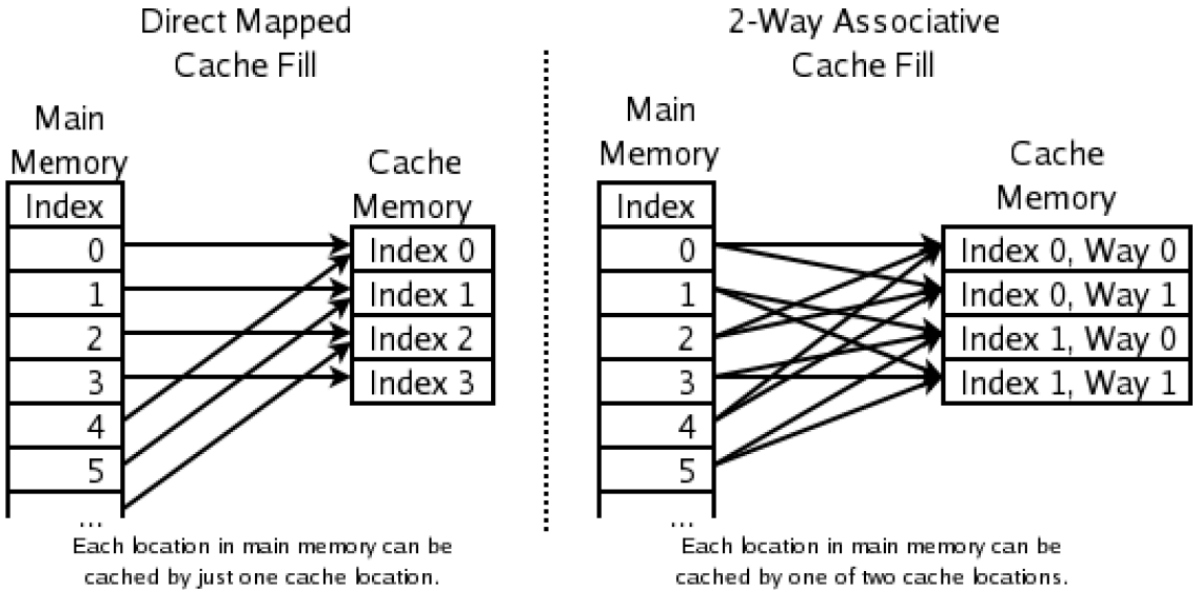
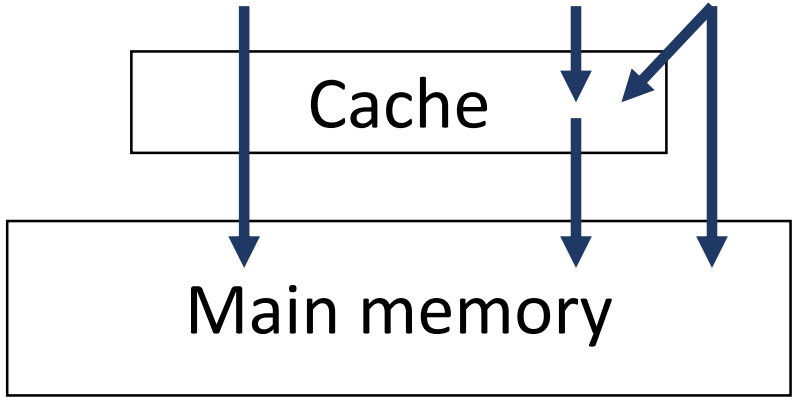


Figure from wikipedia

Cache: Other design choices

- Replacement policy
  - Random, LRU, FIFO, …
  - Each has strengths & drawbacks
- Write policy
  - Write-through, writeback, write-no-allocate
  - Related to coherency management



Implementing cache [High-level design]

- Cache requirements
  - Direct-mapped, single level cache
  - Capacity: 16 words / Line size: 4 words
  - LRU replacement policy
  - Unified cache
  - Write policy (refer to the next slide)

  Should be a part of the CPU (not Testbench)

Implementing cache [Write policy]

- Write policy
  - Write-through cache & write-no-allocate
  - Write-back cache & write allocation
- You can freely choose either write-through or write-back.
- For more detail, We provide example control flows of each write policy. *(Refer to the next slide)*

Implementing cache [Write policy] (cont.)

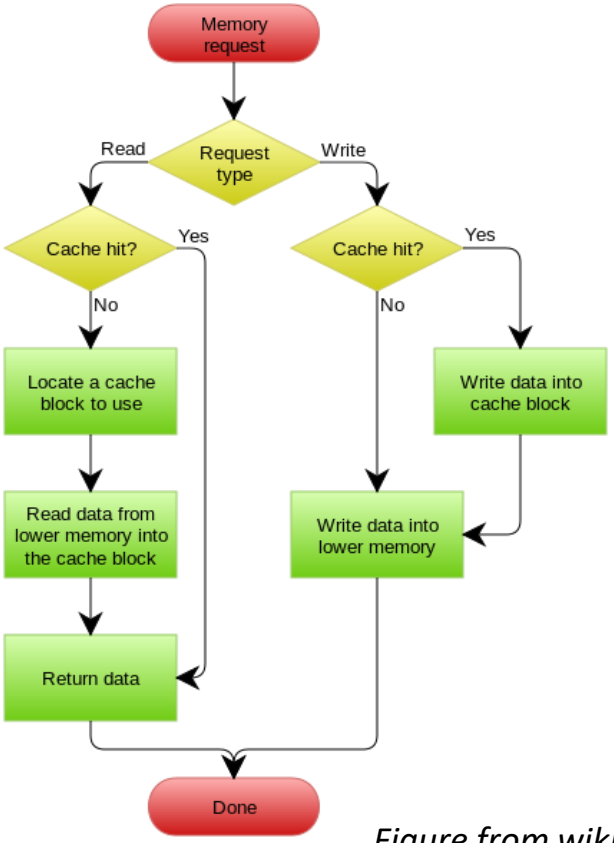


Figure from wikipedia

Write-through // write-no-allocate

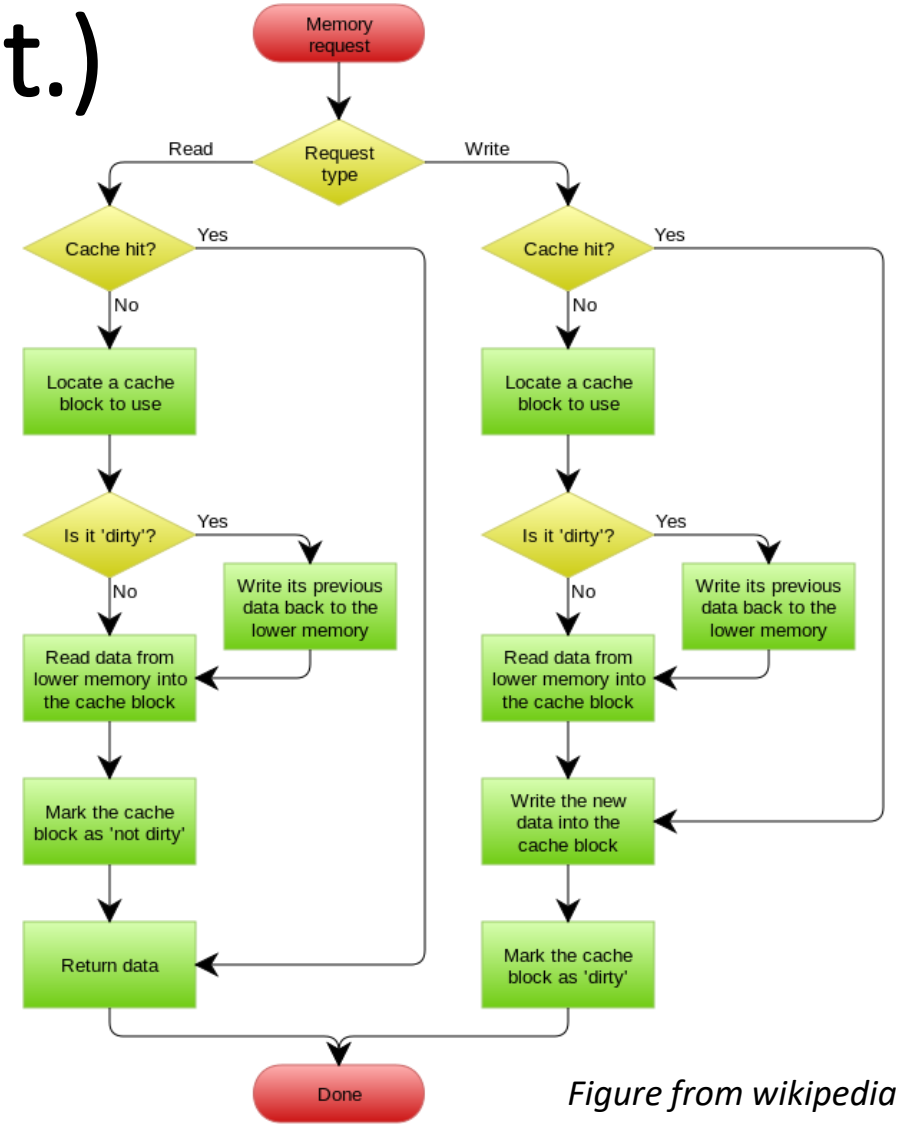


Figure from wikipedia

Write-back // write-allocation

Implementing cache [Latency model]

- Model latencies
  - Two models (Baseline CPU, CPU w/ cache)
- Baseline CPU
  - One memory access fetches **one word** and takes **two cycles**.
- New CPU (w/ cache)
  - Cache hit takes **one cycle**
  - One memory access should fetch **four words into cache** (== one cache line) and take **six cycles**.

You should change your memory module (memory.v) to support above latency models.

Evaluating the performance

- Before doing performance comparison, you should **pass all test cases** we provided.

- In the report
  - Calculate the hit (or miss) ratio
    - Hit ratio = (# of hits) / (# of memory accesses)
  - Compare the performance
    - Baseline CPU vs. new CPU
    - You should use the new latency models.

Summary

- Understand cache
  - What it is
  - How it looks like
  - How it works

- Implement a **direct-mapped cache** on your pipelined CPU
- Evaluate the benefits
  - vs. baseline CPU (w/o cache)